8. TRAINING OF NEURAL NETWORKS IN WEBDEDIP ENVIRONMENT

8.1 OVERVIEW OF ARTIFICIAL NEURAL NETWORKS

The human brain's powerful thinking, remembering and problem solving capabilities have inspired many scientists to simulate its operations using computer modeling. The brain is most complex system comprising of billions of neurons made up of the cell body, the axon and dendrites. The response of neuron depends upon several biological and chemical factors corresponding to the synapses and the receiving neuron. A neuron may fire a signal if magnitude of the signal is strong enough to activate it. The synaptic efficiency or strength is modified to adjust to the received signal. The brain is said to learn when the synapse adjust themselves to receive new signals.

Artificial Neural Networks (ANNs) are computational models, which attempt to mimic the learning function of brain. As brain learns from examples, ANN is trained by presenting it the set of input-output patterns. Because of this, ANN can be used to solve complex problems, where predefined knowledge is not required. In general the most useful properties of neural networks are self-organization, generalization, fault tolerance and massive parallelism.

Artificial neural networks are composed of a set of neurons or processing units, which are connected together by means of connecting weights. ANNs are structured to learn and generalize so that network may learn by continuous adjustment of weights of connections. Typically ANNs consist of input layer to which data is presented for network, hidden layer and output layer where the response of the network to a given input is received. The ANN requires a number of input-output patterns for training. The purpose of training is to adjust a set of initially randomized weights, using a training algorithm, until a non-linear and continuous function representing a mapping space, which includes the training patterns, may be formed. Such networks learn the mapping task by using examples and after learning provides output for unknown instances instantaneously.

The training process is time consuming as it may take many iterations in order that net reaches a desired accuracy. It takes more time particularly when either the size of net is large or the training pattern is large. The training process can be made faster by utilizing parallel nature of the algorithm. In this chapter implementation of training of ANN on distributed processing environment is discussed to speed up the process.

8.2 SELECTED NETWORK AND PROBLEM

Adeli and Park [119] in 1995 employed Counter Propagation Neural network (CPN) in analysis and design of beam and plate problems and demonstrated its superiority over the widely used Back Propagation Neural network (BPN) especially for the problems involving thousands of links. Therefore, CPN is selected in the present work. Further to improve computational efficiency, the greatest scope lies with the parallel nature of the algorithm. This nature enables the algorithm to be implemented on parallel or distributed computing environment. Parallel implementation of multi layered neural networks may be carried out in a number of ways including [89]:

(i) Distribution of the network by dividing the units and or layers amongst the processors;
(ii) Distribution of units by representing each unit with a single processor; and
(iii) Distribution of input-output data patterns among the processor.

If network structure is not large but a large number of training patterns are to be used then third approach is likely to be more efficient.

In recent times due to readily available network of workstations (LAN), application developer can speed up computations using the available computers on a local area network in parallel. As a result, the network of computers can be treated as a virtual parallel machine. The common benefits of distributed applications include resource sharing, performance enhancement, availability, extensibility, and reliability. In the present work for distributed processing, WebDedip environment is used.

To demonstrate the implementation of training of neural network, an application of design of rectangular short and slender column sections is discussed. The

Counter Propagation Neural network [120] which combines Kohonen layer with competitive units that undergo unsupervised learning with the Grossberg layer which is fully connected to Kohonen layer and undergo supervised learning is used. More than 7400 data sets are extracted from the charts given in the design aid SP:16 [121] for the design of rectangular columns. The data set in form of input-output are distributed among the slave computers where training of network is carried out and final weights are collected on master computer. Comparison of time between sequential and distributed version of training is made.

8.3 DATA PROCESSING IN CPN

A counterpropagation network has three-layer architecture as shown in Fig. 8.1. First layer is an input layer; the number of neurons in this layer corresponds to number of inputs. The next layer is taken as a Kohonen layer, which is a hidden layer, and all nodes in this layer are competitive. This layer is trained through winner take all learning algorithm in an unsupervised way. Each neuron of this layer represents an input cluster, so if layer works in local representation, a particular m^{th} neuron's input and the response will be the largest. This neuron is considered as a winner neuron for that cluster. Hence for similar input vectors, belonging to the same cluster activates the same m^{th} neuron among all other neurons of this layer. Further all neurons of this layer are assumed with continuous activation function during the learning. The last layer, which is known as Grossberg layer, is an output layer. All neurons of this layer are fully connected with hidden neurons and are non-competitive. Training of this interpolation layer is supervised where weight vector between winner neuron and output layer is adjusted by Outstar learning algorithm.

In CPN, the weight vector between input instances and Kohonen layer is stabilized first and then convergence of weights between Kohonen and Grossberg layer takes place. Once the unsupervised learning phase is completed and weights are stabilized, interpolation layer starts to learn the desired output. Input data set from the training pattern is presented to the competition layer as a fresh and the winning node j in the competition layer is selected. Then the supervised learning is performed on the connections from winning unit j to all the units in the output layer. The CPN can then learn the given set of vector pairs

(X_1,Y_1), (X_2,Y_2), ... (X_i,Y_i) to associate X vector on the input layer with Y vector at the output layer. The objective of the CPN is to describe the approximate mapping relation between X and Y in to bi-polar binary response for the entire range specified by the set of training instances.

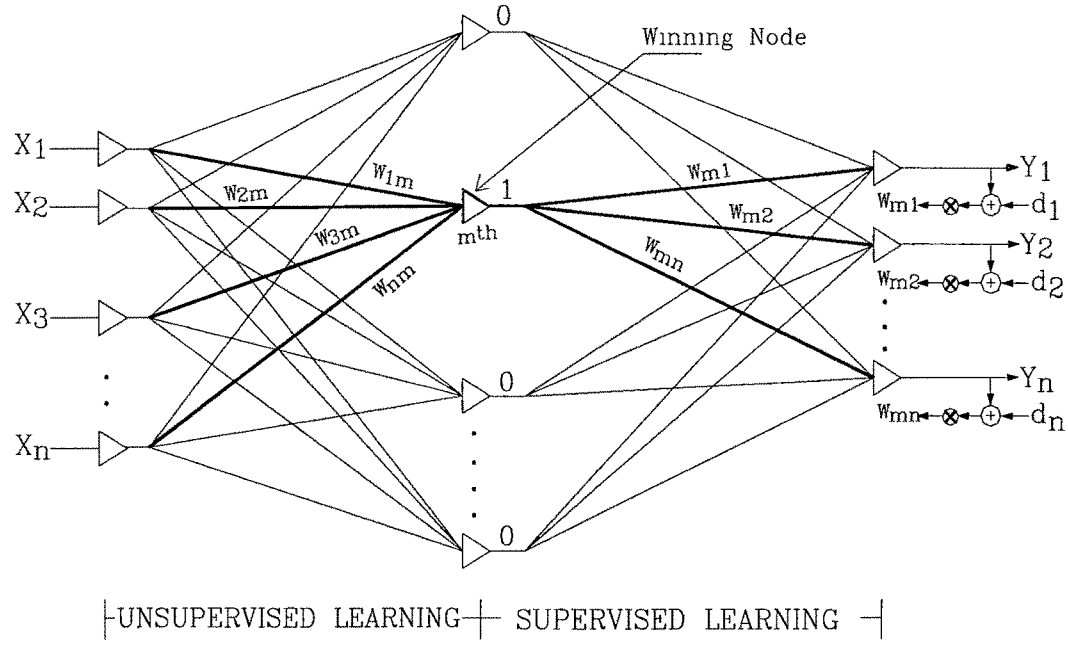

FIG. 8.1 BASIC TOPOLOGY OF COUNTERPROPAGATION NETWORK

The properties of different layers and steps of learning process are as follows:

1. Initialize the network weights between input and hidden layer in proportion to the normalized reflection of the input pattern.

2. Assign the input values of training pattern one by one.

3. Train the weights from input to the Kohonen layer nodes through unsupervised winner take all learning rule. For each iteration, the network is presented with all the input samples of training data sets. Let Wj be the arbitrary weight vector assigned to the links between input nodes and j^{th} node in the competition layer. For each input sample, the nearest node in competition layer is obtained by means of Euclidean distance dj with the help of

$$dj = [\Sigma (X - Wj)]^{1/2} \qquad \ldots (8.1)$$

4. Hold the competition. The node having minimum Euclidean distance is considered as the winner node and its activation is set to 1 while for all others

it is set to 0. Let j^{th} neuron wins the competition, hence output vector Z_j of the competition layer becomes,

$$Z_j = [\ y1\ \ y2\ \ y3\ \ .\ .\ .\ y_j\ .\ .\ .\ yp\ ] = [0\ \ 0\ \ 0\ \ .\ .\ .\ 1\ .\ .\ .\ \ 0\ ] \quad \ldots (8.2)$$

5. Adjust only those weights which are corresponding to the winning node using the following equation for $(n+1)^{th}$ iteration.

$$W_{ji}\ (n+1) = W_{ji}\ (n) + \alpha\ [\ X_i - W_{ji}\ (n)\ ]\ Z_j \quad \ldots (8.3)$$

where α is small positive learning coefficient and for one iteration keep this learning rate constant and as iteration progresses shrink the learning rate. Shrinking in the learning rate as a function of iteration number as mentioned in Eq.8.4 shows comparatively a faster convergence.

$$\alpha = 1/(\ n+1)^2 \quad \ldots (8.4)$$

6. Compute the error term which is an average Euclidean distance of all winning nodes for that iteration.

7. Repeat steps 2 to 6 till desired convergence in error term is achieved.

8. Initialize the random weights between competition nodes and output layer.

9. Assign the training pattern with input and desired output to the input layer.

10. Select the winner node from the competition layer using stabilized weights of unsupervised learning for the input set.

11. Perform supervised learning using a weighted summation function as a transfer function for interpolation layer and adjust the weights between winner node j and all the units in output layer according to the learning rule suggested by Grossberg as,

$$V_{ji}\ (n+1) = V_{ji}\ (n) + \beta\ [\ Y_i - V_{ji}\ (n)\ ]\ Z_j \quad \ldots (8.5)$$

where β = small positive learning constant and is taken in the range of $0 < \beta < 1$.

12. Determine output of the network from the i^{th} node of interpolation layer as

$$Y_i = \Sigma\ V_{ji}.Z_i \quad \ldots (8.6)$$

13. Compute Root Mean Square (RMS) error as an error term for this learning.

14. Repeat the above steps 9 to12 till this learning ensures the desired reduction in the error term i.e. till the output pattern becomes similar to the desired output.

After connection weights are stabilized through winner take all and outstar learning rules, performance of the network is tested using untrained instances. During testing number of winning nodes in competition layer may be set to value more than one depending on the type of the problem. In case of number of winning nodes, during verification stage, their non zero outputs are set such that the winner node associated with the weight vector closest to the given untrained instance has the largest output. However, the sum of the outputs of all winning nodes in the competition layer must remain equal to 1. Actual output is then computed through interpolation of these winning nodes output. If S_w is the set of n winning nodes, output of the winning node Z_j is given by

$$Z_j = \begin{cases} \dfrac{\sum_{k=1}^{n}(d_k)-d_j}{\sum_{k=1}^{n}(d_k) * (n-1)} & \text{if } Z_j \in S_w \\ 0 & \text{if } Z_j \notin S_w \end{cases} \quad \ldots (8.7)$$

where $\Sigma(d_k)$ is the summation of Euclidean distances of first n nearest winning nodes and d_j is the Euclidean distance of corresponding winning node.

8.4 CPN SIMULATED COLUMN DESIGN

The exact design of column section subjected to an axial load and bi-axial bending moment as shown in Fig. 8.2 is extremely laborious. A simplified approach is given in IS: 456 [122] based on the concept of failure surface where the load - moment interaction diagram is assumed to be extended in three dimensions. In this approach also, however, solution requires several steps. Some of the steps are to be executed in parallel with the manual reference of design aid SP: 16 [121] and the design procedure is to be iterated till acceptable solution is obtained. Further for getting optimum design solution several trials are necessary and also picking the values from the available graphs is quite tedious and time consuming.

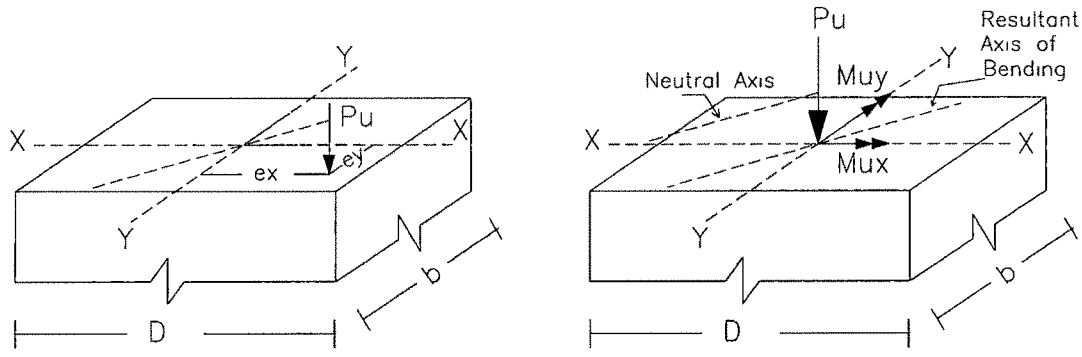

FIG. 8.2 RECTANGULAR SECTION UNDER BI-AXIAL BENDING

Here CPN is used for structural design of column. For training the design network, data is gathered from 24 charts given in SP:16 (Chart Nos. 27 to 50) [121] which cover different grades of steel and concrete. More than 7400 training data sets are extracted from these charts to build a network with four inputs Pu/Fck.bD, *p*/Fck, d'/D and grade of steel Fy and one output Mu/Fck.bD^2 for reinforcement detailing. The typical input-output pattern is shown in Table 8.1.

TABLE 8.1 SAMPLE TRAINING DATA SETS FOR BIAXIALLY LOADED COLUMNS

Sr No	Pu / Fck bD	*p* / Fck	d' / D	Fy MPa	Mu / Fck bD^2
01	0 08	0 02	0 05	415	0 06
02	0 12	0 04	0 05	415	0 095
03	0 2	0 14	0 05	415	0 211
04	0 28	0 24	0 05	415	0 318
05	0 4	0 06	0 05	415	0 09
06	0 52	0 14	0 05	415	0 147
07	0 72	0 18	0 05	415	0 13
08	1 04	0 24	0 05	415	0 088
09	0 08	0 24	0 1	415	0 305
10	0 24	0	0 1	415	0 055
11	0 36	0 26	0 1	415	0 29
12	0 56	0 04	0 1	415	0 005
13	0 72	0 1	0 1	415	0 02
14	0 92	0 24	0 1	415	0 125
15	0	0 06	0 15	415	0 078
16	0 04	0 22	0 15	415	0 237
17	0 12	0 1	0 15	415	0 136
18	0 16	0 26	0 15	415	0 272
19	0 24	0 14	0 15	415	0 169
20	0 28	0 08	0 15	415	0 114

Sr No	Pu / Fck bD	*p* / Fck	d' / D	Fy MPa	Mu / Fck bD^2
21	0 32	0 24	0 15	415	0 243
22	0 36	0 18	0 15	415	0 188
23	0 4	0 12	0 15	415	0 127
24	0 44	0 06	0 15	415	0 069
25	0 52	0 2	0 15	415	0 17
26	0 6	0 18	0 15	415	0 138
27	0 68	0 2	0 15	415	0 136
28	0 92	0 16	0 15	415	0 019
29	0	0 2	0 2	415	0 189
30	0 08	0 08	0 2	415	0 105
31	0 2	0 12	0 2	415	0 135
32	0 28	0	0 2	415	0 05
33	0 32	0 16	0 2	415	0 15
34	0 36	0 1	0 2	415	0 102
35	0 56	0 1	0 2	415	0 07
36	0 6	0 1	0 2	415	0 058
37	0 76	0 2	0 2	415	0 1
38	0 88	0 2	0 2	415	0 071
39	1 04	0 22	0 2	415	0 04
40	1 24	0 26	0 2	415	0 015

The counterpropagation initialization function of the program initializes all weight values of the competition layer first. Every component W_{ij} of every competition layer node is then assigned a random value depending upon the input patterns. Next the weights of the interpolation layer are initialized to a random value depending upon the output patterns. A CPN with 4-7440-1 topology is selected, which converges in 30 cycles for unsupervised learning and in 25 cycles for supervised learning.

8.5 IMPLEMENTATION OF CPN TRAINING IN WEBDEDIP ENVIRONMENT

To implement the training process over distributed computing environment, the total number of training sets need to be divided in to number of sets (tasks), which can run on different computers, and communication between these tasks can be carried out using intermediate files. The process of distributed training of ANN using counter propagation algorithm is divided into three tasks. The first task (DATADIST) divides the total input-output patterns in to the number of sets equal to number of computers available on network. Theses data sets are transferred to different computers. The second task (CPNTRAIN) is the training of network for allotted data set on different computers. The training of network consists of finalization of weights between various layers as described in the earlier section and are written in the files. These files are transferred to one computer. The third task (CPNFINAL) combines the finalized weight files from all computers to have common weight file which can be used for further testing. The interface among different tasks is carried out using intermediate files. All the tasks are then inter linked using DEDIP GUI to configure the application for parallel processing.

Figure 8.3 shows screen shot depicting the required interdependency. The DEDIP GUI is used to provide the information about remote node on which the process is to be executed. In this application DATADIST and CPNFINAL tasks are carried out in sequence while CPNTRAIN is carried out in parallel depending on number of computers. To achieve better performance, the time for the task running in parallel should be kept larger than that of sequential process.

The training of network is carried out using different number of computers on the network. After configuration of application and successful run of application,

WebDedip gives the summary of application indicating the status of various processes i.e. node number (IP address), start time, end time. The time required for computation and communication can be obtained from this summary when process is distributed over number of computers. In the present implementation network of Pentium – IV computers running at 1.8 GHz with 256 MB RAM connected through 100 Mbps Ethernet was used.

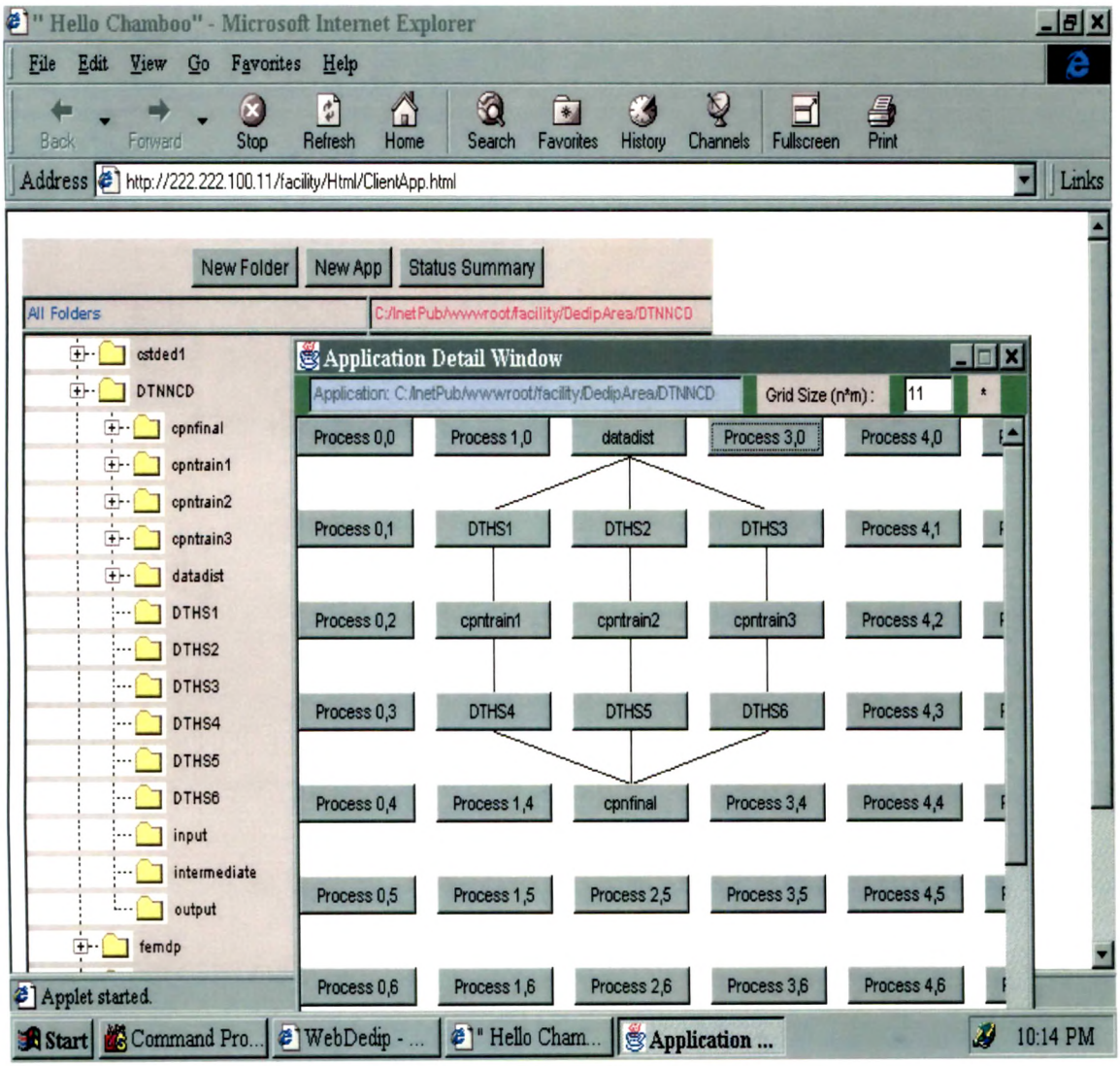

FIG. 8.3 DEDIP GUI TO CONFIGURE THE APPLICATION ON 3 COMPUTERS

Table 8.2 shows the time taken by various tasks when distributed over different number of computers. It also includes total computation time and communication time and its comparison with sequential time. From the observation of time the speed-up is calculated. Ideal speed-up is equal to number of computers used. But due to time involved in communication of data

between the computers, observed speed-up is lower than ideal speed-up. The comparison of computation time and communication time is shown in Fig. 8.4. The speed-up observed is shown in Fig. 8.5.

TABLE 8.2 COMPARISON OF TIME REQUIRED FOR TRAINING

No. of Computers	Name of Task	Time in Sec	Total computation time	Communication Time in Sec	Observed Speedup	Efficiency (%)
1	-	-	1470	0	1	100
2	DATADIST	12	773	100	1.68	84.19
	CPNTRAIN	748				
	CPNFINAL	13				
3	DATADIST	15	533	110	2.29	76.21
	CPNTRAIN	503				
	CPNFINAL	15				
4	DATADIST	17	415	120	2.75	68.69
	CPNTRAIN	380				
	CPNFINAL	18				
5	DATADIST	18	343	135	3.08	61.51
	CPNTRAIN	305				
	CPNFINAL	20				

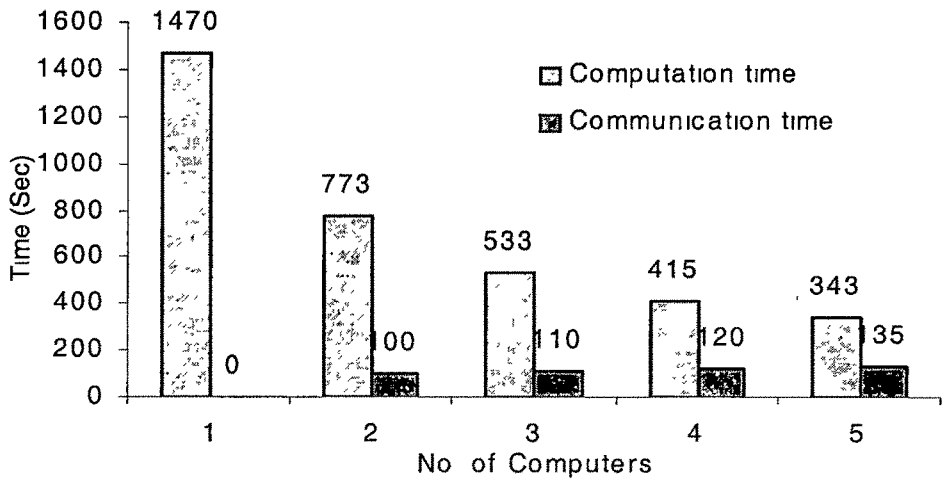

FIG. 8.4 COMPARISON OF COMPUTATION AND COMMUNICATION TIME

In the present work computational load is distributed in the beginning of the process, which is known as static load balancing. As all computers are of same

configuration, equal number of training patterns are distributed to number of computers. Adjustment of load during processing, known as dynamic load balancing, is not possible in this implementation because depending on number of input-output patterns network topology is decided in the beginning of the process, which can not be changed subsequently.

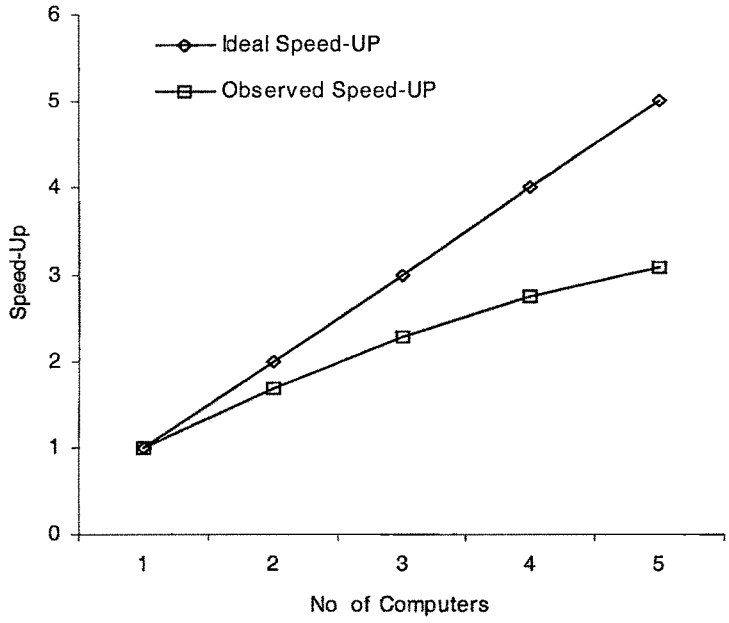

FIG. 8.5 COMPARISON OF SPEED-UP

8.6 CLOSING REMARKS

The training is the most compute intensive part of ANN and to achieve speed in training parallel or distributed computing can be used. In the present Chapter, distributed training was implemented over the Local Area Network using WebDedip. Use of WebDedip simplified the distributed application development. For training of network Counter Propagation learning algorithm was used. The distributed training was demonstrated through an example of design of rectangular R.C.C. column subjected to an axial load with biaxial bending. To implement distributed processing, number of input-output patterns were distributed over different number of computers for training. The efficiency of order 60 – 80% was observed during distributed training of ANN.